Motivation for new Device class

- Create a common superclass with a rich interface
- Allow enumeration of instantiated devices (e.g. “Give me all the motors”)
- Allow exploration of device properties through a generic interface (including properties specific to the specialization)
- Provide infrastructure for device thread safety

Devices

Devices include the following:

- Attributes
  - Static configuration information
- Parameters
  - Dynamic configuration information
- Telemetry
  - Data produced by device

Device Attributes

- Device attributes are static information that is typically required to instantiate a device
  - Name
  - Hardware connection info (e.g. address)
  - Mechanical properties (e.g. ticks per radian)
  - Capabilities (e.g. can measure current)
  - Limits (e.g. maximum velocity)
  - Requirements (e.g. requires calibration)
- Device attributes do not change at runtime

Device Parameters

- Device parameters are configuration information given to the device which may change at runtime
  - Operational parameters (e.g. % of maximum speed for manipulator motions, servo loop rate)
  - Externally controlled goals (e.g. charging profile for a battery charger)
  - Telemetry update rate

Device Telemetry

- Device telemetry is information produced by the device at runtime
  - Sensed information (e.g. measured position or voltage)
  - Internally controlled goals (e.g. in response to command)
  - Status (e.g. power on/off, is calibrated, in fault)
  - Future: commands and their completion

Device Methods

Identification

- string get_name()
- string get_type_name()
- string get_ancestry()
  - Returns a colon-separated list of ancestor typenames
- string get_impl_type_name()
- string get_impl_ancestry()

Device Methods, contd.

Status

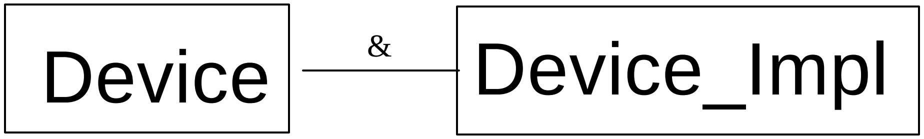
- get_status()
- on(), off(), is_on()
- initialize(), is_initialized()
- calibrate(), is_calibrated()
- in_fault(), clear_fault()

Device Methods, contd.

Attributes, parameters, telemetry access:

- get_attribs
- get_params
- get_latest_telemetry
  - Get most recent cached data, don’t put the device to extra effort
- update_telemetry
  - Force an update

Interface/Implementation isolation



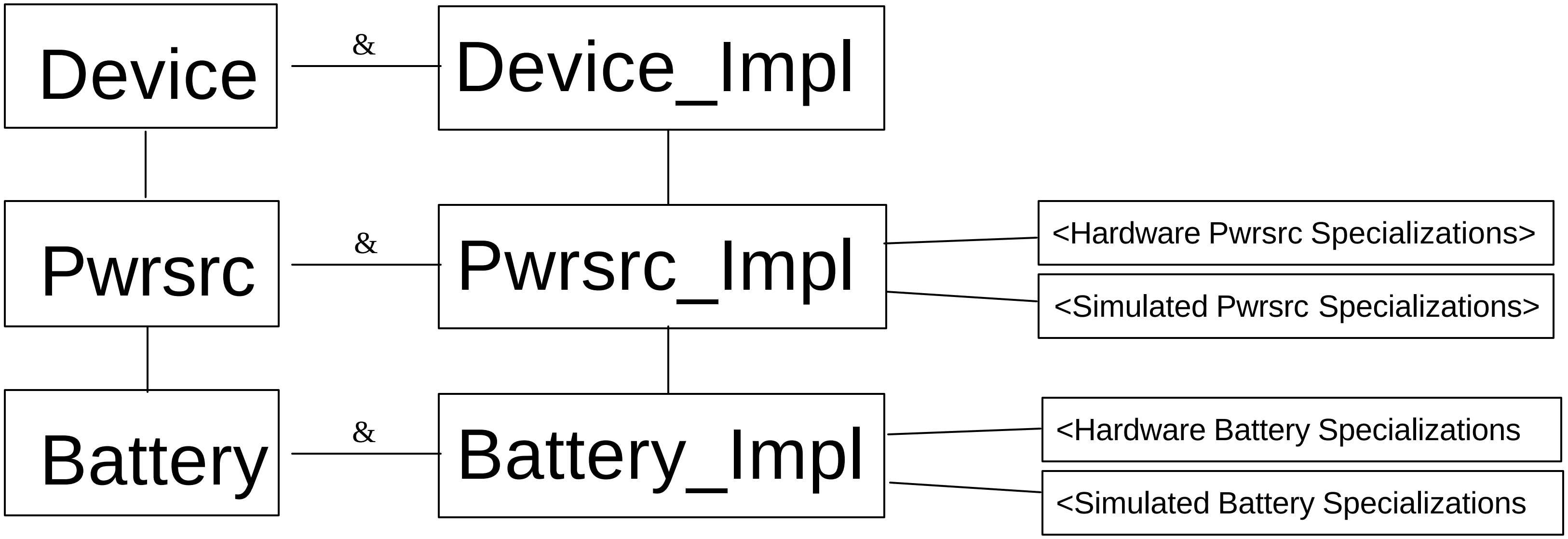
Device

&

Device_Impl

- Devices use the bridge pattern to isolate interface specialization from implementation specialization.
- Device is superclass for interface specialization
- Device_Impl is superclass for implementation specialization
- Users only interact with Device and its specializations, never with impls
- Device holds a reference to a Device_Impl
- Device constructor takes a Device_Impl and a bool for whether or not to delete it when the Device is destructed

Device Inheritance Hierarchy

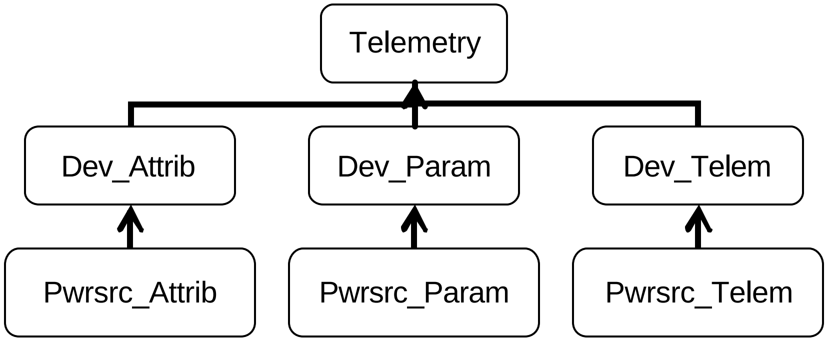

Device
&
Device_Impl
Pwrsrc
&
Pwrsrc_Impl
<Hardware Pwrsrc Specializations>
<Simulated Pwrsrc Specializations>
Battery
&
Battery_Impl
<Hardware Battery Specializations
<Simulated Battery Specializations

- Each kind of device defines both a Device subclass and an associated Device_Impl subclass
- Hardware specialization is done by subclassing the Impl for the appropriate kind of device

Telemetry class

- Telemetry is base class to represent time-stamped data
- Has interfaces for:
  - Type name/ancestry
  - Serialization/deserialization
  - Cloning
    - Virtual method overridden by each subclass to create a new object of its own type using its copy constructor
    - Allows copying all data fields without requiring advance knowledge of or dependence on Telemetry subclasses

Dev_Attrib, Dev_Param, Dev_Telem



- Device_Impl has pointers to objects inheriting Dev_Attrib for attributes, Dev_Param for parameters, and Dev_Telem for telemetry, all of which inherit from the Telemetry class
- The actual instantiations of these objects will be appropriate to the Device_Impl subclass
  - For example, Pwrsrc_Impl would instantiate Pwrsrc_Attrib, Pwrsrc_Param, and Pwrsrc_Telem which add fields specific to Pwrsrc
- This allows superclasses to access and modify the fields they know about in the same object that contains the more specific information
- This allows specific information to be accessed through generic interfaces at the Device level

Device Methods Revisited

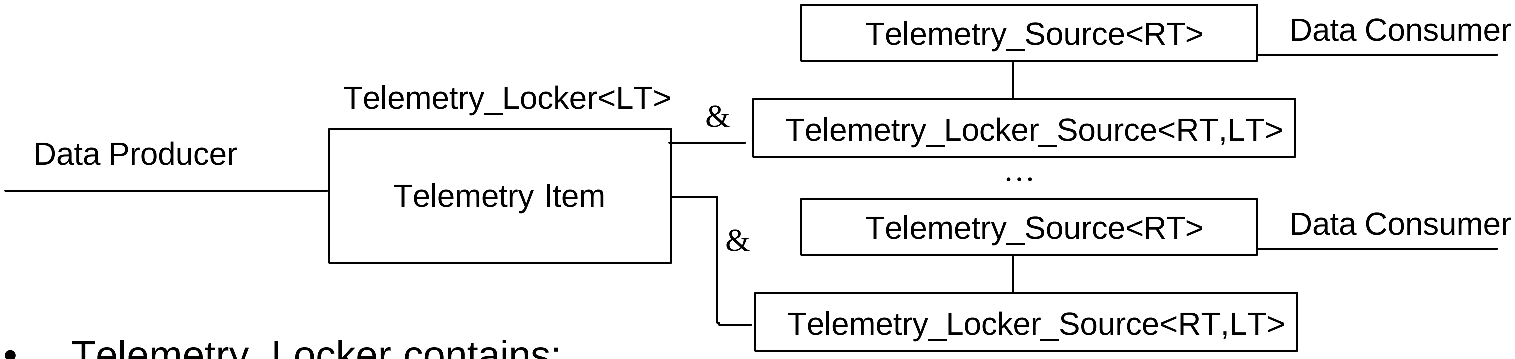
Device-level accessors are templatized for convenient use of subclasses. Attribs used as example, analogous methods for params and telem.

- template <T> bool get_attribs(T &attribs)
  - Copies attributes into attribs
  - T can be the actual instantiated class, or any ancestor. If not either of these, returns false and does not set attribs.
  - Additional subclass data beyond T is lost
  - Example: Instantiated type is Battery_Attrib, user passes in Pwrsrc_Attrib, succeeds but loses Battery-specific data
- template <T> bool get_clone_attribs(auto_ptr<T> &attribs)
  - Similar to above, but internally calls clone, which causes a malloc but gets all the subclassed data.
  - The user has to free the data, which is facilitated and made explicit by use of auto_ptr
  - If attribs already contained non-NULL pointer, is automatically deleted by assignment inside get_clone_attribs

Telemetry Source<T>

- Provides abstract interface to a source of data, abstracting away dependence on the data producer
- Templatized on the type of Telemetry provided
- Supports:
  - get_next_telemetry, with optional timeout
  - get_latest_telemetry and get_earliest_telemetry, with optional newer_than time and timeout
- Contains parameters for minimum and maximum intervals, with set and get methods
- Remembers timestamp of last data returned to user, with set and get methods
- Blocking semantics if timeouts not used
- One-to-a-customer, should not be shared

Telemetry Locker



- Telemetry_Locker contains:
  - An auto_ptr<LT> pointing to a data object exclusively held by the locker
  - A lock
  - A condition variable
- The data producer writes to the locker when it has new data
  - The write will either clone an item provided as a const LT &, or take ownership of an item provided as an auto_ptr<LT> &
  - The item in the locker will be of type LT, or a subclass of LT
  - Writing to the locker triggers the condition variable, waking any blocking consumer threads
- Consumers access the data through a specialization of Telemetry_Source<RT> which has a reference to the locker
  - The Telemetry_Locker_Source is templatized on both on LT, the type explicitly in the locker, and RT, the type returned by the Telemetry_Source.
  - So long as RT matches or is an ancestor of the type of the item actually held in the locker the cast will succeed, so RT may be more specific than LT.
  - All blocking reads in Telemetry_Locker_Source are implemented as condition variable waits, so any threads waiting on these will be woken up when an item is written
  - Many Telemetry_Locker_Source instantiations can refer to the same Telemetry_Locker

Device Telemetry Sources and Lockers

- Device_Impl contains two telemetry lockers:
  - Telemetry_Locker<Dev_Param> _dev_plocker
  - Telemetry_Locker<Dev_Telem> _dev_tlocker
- When new params are set or telemetry is updated new data is written to the lockers
- Device provides generic interfaces to request telemetry sources:
  - get_telemetry_source_names
  - get_telemetry_source(name)
  - get_dev_telemetry_source
- Device specializations provide specialized versions for their specific Dev_Telem type (e.g. get_battery_telemetry_source returns a Telemetry_Source<Battery_Telem>)
- These calls create a new Telemetry_Locker_Source attached to one of the Device_Impl lockers, and return the result in an auto_ptr to indicate that the caller owns the object and needs to free it.

Telemetry_Logger<T>

- Telemetry_Logger is a superclass for objects which log Telemetry_Source<T>
  - It provides an add_source call which takes ownership of the source, and can associate a name with it
  - It starts a thread for each source, and uses the blocking get_next_telemetry call, so no polling is done
  - Methods are provided to get the names and states of sources being logged, and to start and stop logging of sources, either individually or all at once.
- Specializations override the virtual _send_item call to take appropriate action.
  - Examples have been implemented for c++ streams, Chris Urmson's socket layer, and a CORBA link